A Support Vector Regression (SVR)-Based Latency Model for Network-on-Chip (NoC) Architectures

Zhi-Liang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, *Senior Member, IEEE*, Diana Marculescu, *Fellow, IEEE*, and Radu Marculescu, *Fellow, IEEE*

***Abstract*—In this paper, we propose SVR-NoC, a network-on-chip (NoC) latency model using support vector regression (SVR). More specifically, based on the application communication information and the NoC routing algorithm, the channel and source queue waiting times are first estimated using an analytical queuing model with two equivalent queues. To improve the prediction accuracy, the queuing theory-based delay estimations are included as features in the learning process. We then propose a learning framework that relies on SVR to collect training data and predict the traffic flow latency. The proposed learning methods can be used to analyze various traffic scenarios for the target NoC platform. Experimental results on both synthetic and real-application traffic demonstrate on average less than 12% prediction error in network saturation load, as well as more than 100× speedup compared to cycle-accurate simulations can be achieved.**

***Index Terms*—Latency, learning, network-on-chip (NoC), queuing theory, support vector regression (SVR).**

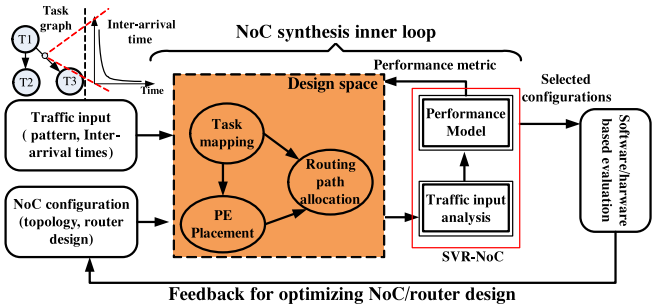

Fig. 1. NoC performance model used in the synthesis inner loop for design space exploration [3], [4]: as highlighted in the box, each feasible task mapping, PE placement, and route allocation lead to a design choice, which needs to be evaluated using an efficient performance model.

I. Introduction

NETWORK-ON-CHIP (NoC) architectures have been proposed as high performance and scalable methods to handle the complicated on-chip communications [1], [2]. Fig. 1 shows a typical design flow of an NoC-based system-on-chip (SoC) which involves several steps such as task allocation, processor mapping and packets routing; each of the synthesis steps can produce a variety of design choices [3], [4]. Performance analysis methods are therefore needed to evaluate whether the chosen architecture corresponds to a better design [3], [4]. Compared to simulations, fast NoC performance models are more widely adopted in the system synthesis inner loop due to their better tradeoff between the accuracy and speed [5], [6].

Among various metrics, latency is one of the most critical measures since it characterizes the system operation speed [6]. In order to derive a latency model, much of the previous work is based on the queuing theory, which treat each input channel in the NoC router as an $M/M/1/\infty$ [7], [8] or $M/G/1/N$ [5] queue. The M notation in such queues assumes either the traffic interarrival time follows a Poisson distribution at the injection sources [3], [5] or the service process satisfies the memoryless property [7]. However, it has been observed that in many NoC systems, the traffic interarrival time, as well as the service time distributions may not follow exactly the above assumptions [9]. Consequently, the accuracy of the queuing theory-based models is limited.

In this paper, we develop an efficient NoC latency model by using a machine learning-based modeling technique. More specifically, we first propose a queuing delay model that combines the advantages of several previous models such as [3], [6], [10], and [11], and uses two equivalent queues. In order to further improve the accuracy, we then propose a support vector regression (SVR)-based NoC latency model. In SVR-NoC, the delay predictions obtained from the proposed analytical model are included as part of the features in the learning process. In this way, the learning model allows us to analyze the flow delays based on not only the queuing predictions but also the simulation results of the training samples.

Manuscript received November 26, 2014; revised April 7, 2015; accepted July 2, 2015. Date of publication August 28, 2015; date of current version February 24, 2016. This work was supported in part by the Hong Kong Research Grants Council under Grant GRF619813, in part by the HKUST Sponsorship Scheme for Targeted Strategic Partnerships, and in part by the U.S. National Science Foundation (NSF) under Grant CNS-1128624. The work of P. Bogdan was supported by the NSF under Grant 1453860 and Grant 1331610. This paper was recommended by Associate Editor L. P. Carloni.

Z.-L. Qian was with the Hong Kong University of Science and Technology, Hong Kong. He is now with Shanghai Jiaotong University, Shanghai, China (e-mail: qianzl@sjtu.edu.cn).

D.-C. Juan was with Carnegie Mellon University, Pittsburgh, PA 15213 USA. He is now with Google, Mountain View, CA 94043 USA (e-mail: dacheng@alumni.cmu.edu).

P. Bogdan is with the University of Southern California, Los Angeles, CA 90089 USA (e-mail: pbogdan@usc.edu).

C.-Y. Tsui is with the Hong Kong University of Science and Technology, Hong Kong (e-mail: eetsui@ust.hk).

D. Marculescu and R. Marculescu are with Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: dianam@ece.cmu.edu; radum@ece.cmu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2474393

The rest of this paper is organized as follows. In Section II, we review the previous work. In Section III, we present our proposed queuing-theory-based latency model. Section IV details the learning-based latency model. The experimental results are shown in Section V. Finally, Section VI concludes this paper.

II. Related Work

For NoC average latency evaluation, queuing theory-based models are most widely used. Most early work assumes Poisson arrival time distribution or memoryless packet service time distribution. For example, in [7], the $M/M/1/\infty$ queue model is used to estimate the flit transfer delay over the link. Later, several other works have generalized the arrival or service time distributions. In [11], an analytical queuing model based on $M/G/1/K$ queue is proposed to deal with the finite size of the input buffers in local area networks. More precisely, this approach extracts a single equivalent queue to model the total waiting time of a link. However, for the case of NoCs with a constant packet size (e.g., [12]), usually the ratio of the packet size L and buffer depth B is a fraction. Therefore, it is difficult to determine the equivalent queue capacity in this queuing model (as discussed later in Section III). In [6], an NoC latency model is proposed based on the fixed-priority $G/G/1/\infty$ queue. This model captures the bursty traffic arrival process with a two-state Markov-modulated Poisson process [15]. It also uses the mean and variance of packet length to capture arbitrary packet size distributions. However, this approach targets a specific priority-based router architecture, while many NoC routers may utilize a more fair arbitration scheme such as the round robin [8]. In [14], an $M/M/m/K$ queue-based model is proposed to analyze the delay of heterogeneous NoCs. This approach assumes negligible flit buffers (i.e., a single flit buffer) in each router such that when a packet arrives at the destination processing element (PE), its tail flit has not been issued from the source yet [14]. In [5], an $M/G/1/K$ queue-based model is proposed. However, this approach assumes that the unit of the buffers is dictated by packets instead of flits; therefore a single input port buffer can store several (up to N) packets at the same time. This may not be the case for NoCs whose buffers are small (e.g., only a few flits) [6].

On the other hand, machine learning methods have been widely used in the domains of pattern recognition and artificial intelligence [16], [17]. In NoC design community, machine learning has been used on improving the area/power modeling accuracy. In [18], the multivariate adaptive regression splines technique is used to develop a nonparametric power and area regression model for NoC routers. For latency performance analysis, Qian *et al.* [19] proposed to learn the latency model solely from the simulation data. One limitation of learning from simulation results is that the accuracy highly depends on the similarity between the test cases and training samples. On the other hand, it is difficult to contain all possible traffic patterns in the training stage.

To improve the state-of-the-art models, in this paper, we use the learning techniques in a more efficient way. More specifically, we first propose a new queuing theory-based NoC latency model and then refine this model via learning from simulations. For clarity purposes, in Table I, we summarize and compare our proposed queuing and learning model against several other representative latency models. As shown in Table I, our proposed model offers high flexibility for router architectures exploration. In summary, this paper brings the following new contributions.

1) We propose a new queuing-theory-based latency model which uses two queues in each link for calculating the contention delay and transfer time, respectively.
2) In addition to the proposed queuing model, we propose and develop a learning-based method for NoC latency analysis. We discuss thoroughly how to select various features during the training process, as well as the procedures of applying the learning model.
3) We show the accuracy and scalability of the proposed SVR-NoC model by using both synthetic and real-application traffic patterns.

TABLE I
Summary and Comparison of NoC Latency Models [13]

	Queuing theory based analytical models					**Learning based**
Models	[7]	[3], [5]	[6]	[14]	Proposed queuing	SVR-NoC
Queue	M/M/1/∞	M/G/1/K	G/G/1/∞	M/M/m/K	G/G/1/K	N/A
	Queuing characteristics					
Arrival	Poisson	Poisson	General	Poisson	General	General
Service	Memoryless	General	General	Memoryless	General	General
Traffic pattern	Arbitrary	Arbitrary	Arbitrary	Arbitrary	Arbitrary	Arbitrary
	NoC configurations					
Buffer size	1 flit	K packets	B flits	1 flit	B flits	B flits
L/B [a]	$\gg 1$	< 1	Arbitrary	$\gg 1$	Arbitrary	Arbitrary
Arbiter	Round-robin	Round-robin	Fixed priority	Round-robin	Round-robin	Round-robin

[a] L is the packet length, B is the buffer depth

III. Queuing-Theory-Based Analytical Model

A. Basic Assumptions and Notations

The NoC latency model is used to evaluate a specific design choice in Fig. 1 (i.e., a combination of mapping, placement, and routing solutions). Therefore, the source and destination PE addresses of each flow f are known inputs. Also, similar to [20], we assume that the packet interarrival times of flows have been characterized using a general exponential (GE) distribution [10] (reviewed later in Section III-C). To guarantee deadlocks will not occur at run time, we assume *XY* routing is used [21]. Other deadlock-free and deterministic routing schemes can also be adopted. As in [12] and [22], we assume the size of all flow packets is fixed to be L (flits). Of note, the proposed fixed packet size model can be extended to address general packet length distributions as done in [11] and [23]. Specifically, for a general packet length distribution, we can first compute the delay metrics using the proposed model under a certain packet size. Then the overall service/waiting times are obtained by integration of the results with the packet length probability density function [11], [23]. In the learning framework, the general packet length distribution can also be modeled by adding new features such as the means and variances of packet lengths into the feature vectors. However, more

TABLE II
PARAMETERS AND NOTATIONS IN OUR NoC LATENCY MODEL

Parameters	Description
H	Number of pipeline stages during the flit transmission (including the link traversal stage)
L	Fixed packet length (flits)
B	Buffer size at each input port (flits)
$h_{s,d}$	Overall flow contention delay for packets routing from source PE s to target PE d
$\eta_{s,d}$	Overall flit transfer time for packets routing from source PE s to target PE d
$l_{a,b}$	The link which connects routers a and b
P_f	Set of links that are located in the routing path of flow $f_{s,d}$
F_l	Set of flows that traverse link l
$d(f,l,i)$	The i^{th} downstream link of channel l in the routing path of flow f
λ_f	Mean packet arrival rate (packets/cycle) of the flow f
C_f^2	Squared coefficient of variation (SCV) of the packet arrival process of flow f
C_l^2	Aggregated SCV of the packet inter-arrival times at link l
R_l^2	Aggregated SCV of the packet service times at link l
λ_l	Mean packet arrival rate (packets/cycle) at link l
h_l	Contention delay of a header flit before being granted the use of the link l
η_l	Delay of sending a flit to the front of the buffer in link l
s_l^f	Mean service time for packets from flow f passing through the link l
s_l	Aggregated mean service time for packets from any flows passing through the link l
Pb_l	Blocking (full) probability of the buffer at the link l
v_s	Waiting time at the source PE node s
L_f ($L_{s,d}$)	Average latency of flow $f_{s,d}$ (cycles)

simulations will be needed to capture the samples with different packet sizes. To facilitate the discussion, the symbols in Table II are used in this paper, most of which follow the conventions and definitions in [3], [6], [11], [14] and [23].

B. Overview of NoC End-to-End Delay Formulation

The end-to-end delay $L_{s,d}$ of a specific flow $f_{s,d}$ from source node s to destination d [shown in Fig. 2(a) and (b)] consists of the time needed to wait at the source PE, as well as the time taken to traverse the network [3], [14]. Similar to the terminology defined in [14], $L_{s,d}$ needs to consider three parts: 1) the queuing delay at the source PE s (v_s); 2) the overall flit transfer time ($\eta_{s,d}$); and 3) the channel allocation delay (or namely flow contention delay $h_{s,d}$). In summary, $L_{s,d}$ can be expressed as [11], [14]

$$L_{s,d} = v_s + \eta_{s,d} + h_{s,d}. \tag{1}$$

In order to calculate the overall contention delay $h_{s,d}$, we need to consider the contentions at every link l in the routing path of flow f [3], [14] [Fig. 2(b) shows the links used for the flow $f_{0,8}$] and therefore: $h_{s,d} = \sum_{l \in P_f} h_l$, where h_l is the waiting time of a header flit to be allocated the link l after contending with other flows; P_f represents the routing path of f.

If the time to transmit a flit over the link l is denoted as η_l, then the overall flit transfer time $\eta_{s,d}$ of a packet from s to d can be rewritten as [14]: $\eta_{s,d} = \sum_{l \in P_f} \eta_l + (L-1)$, where the first term denotes the transmission time of header flits along the links in P_f and the second-term approximates the serialization delay of the body and tail flits at the destination [3]. The notations of the queuing delay v_s, h_l, and η_l are also illustrated in Fig. 2.

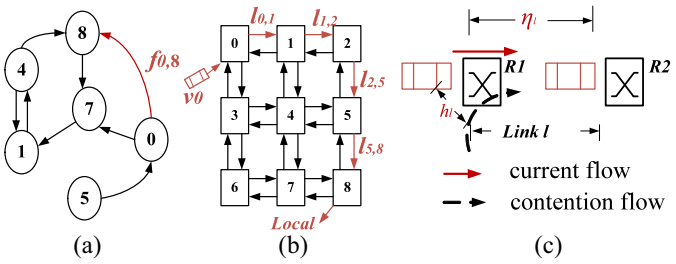

Fig. 2. Example showing the flow delay. (a) Core communication graph (CCG) [3], [6]. (b) Links involved to calculate the flow delay from source 0 to destination 8 ($f_{0,8}$). (c) Contention time h and transfer time η for the link connecting R1 and R2 [11], [14].

From (1), when calculating the flow latency $L_{s,d}$ of a specific flow f, the two delay values (i.e., η_l and h_l) at each link of flow f, need to be obtained. Due to the flow control between neighboring routers, there exist dependencies among the links [11]. For example, in Fig. 2, flows $f_{0,8}$ traverses both links $l_{0,1}$ and $l_{1,2}$. Consequently, the waiting time of link $l_{1,2}$ needs to be calculated first and then to be used in the subsequent derivation of the service time for link $l_{0,1}$.

In order to obtain the link dependencies and determine the order of links for analysis, Kiasari *et al.* [6] used a method to index the link from the sink toward the source channel, while [5] sorts the link channels based on the priorities of dimensions in *XY* routing. More generally, for arbitrary topologies and routing algorithms, a link dependency graph (LDG) can be built first and the topological sorting algorithm is applied on the LDG to properly order the links [11], [13], [23], [24]. In [13], we have presented our algorithm to build the LDG by checking every flow f in the application. An edge is added between two vertices (i.e., links) in LDG if the flow f traverses these links in sequence during the routing. The link order used for analysis is then obtained by applying the topological sort algorithm [25], [26] on the LDG as in [11].

C. Input Traffic Model

Many applications in NoCs have bursty arrival patterns [9]. Therefore, we use the GE distribution [10], [20] to model the arrival traffic at the source PEs and the links. Under the GE distribution, the cumulative distribution function (CDF) of the packet interarrival time T is given by [10] and [20]: $F(t) = \text{Prob}(T \le t) = 1 - \epsilon e^{-\epsilon\lambda t}$, $t \ge 0$, $\epsilon = 2/(1+C^2)$, where (λ^{-1}, C^2) are the mean and squared coefficient of variation (SCV) of T. The GE packet generation process can be considered as a special case of phase-type point process [27], [28]. With a probability $(1-\epsilon)$, a packet is generated directly after the previous packet; otherwise it will experience a Markovian interarrival time before being issued [10], [20]. Therefore, the burstiness of packets is due to the continuous arrivals through the direct branch [10], [20]. By incorporating the second moment (e.g., SCV) in the queuing derivation, the latency model enables the evaluation of the NoC performance under a bursty traffic pattern [6].

D. NoC Router Modeling

In this section, we present the techniques to estimate three key components of the wormhole (WH) analytical models,

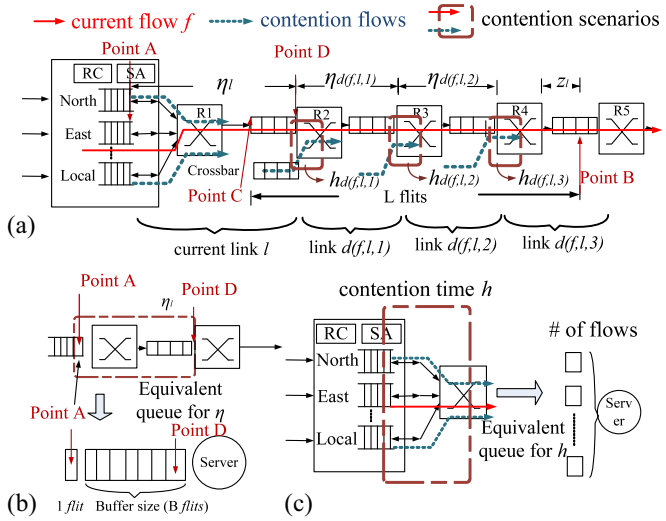

Fig. 3. (a) Illustration of the channel service time of link l [11], [20]. The $d(f, l, j)$ link represents the jth downstream link with respect to current channel l in flow f's routing path. (b) Equivalent queuing system extracted for calculating η_l. (c) Equivalent queuing system extracted for calculating contention delay h_l [11], [14].

i.e., the flit transfer time η_l, the path contention time h_l, and the source waiting time v_s. In [11], [20], and [23], for each link, the overall waiting time (i.e., $\eta_l + h_l$) is first calculated by using an equivalent queueing system. Then, in order to calculate the service time of upstream links, individual η_l and h_l values are computed using the results (e.g., the blocking probability) from that queuing system of link l. However, if the packet size L is larger than the buffer depth B, then the buffer depth to packet size ratio (B/L) is a fraction and smaller than 1. In this case, it is difficult to determine the equivalent capacity of the queue. Therefore, in this paper, we use two queuing systems to calculate η_l and h_l, respectively. Of note, the first queue is used to calculate η_l whose capacity is measured in units of flits. This queue models the time for a flit to move to the head of the buffer and the customers are each individual flits in the packet. For the second queue which is used to calculate h_l, since it models the waiting time of packets from different flows to be granted the use of the output link, the customers in this queue are defined as the packets. Consequently, the queue capacity is measured as the number of different packets that contend for the channel [11], [14]. In the following, we discuss how we use the two queue models to obtain these delay components.

1) Flit Transfer Time Calculation: The flit transfer time η_l of a link l is defined as the time taken for a flit to leave the front of the buffer in the upstream node and arrive at the head of the buffer in the channel l after being allocated that link [14]. Fig. 3(a) illustrates this timing concept. It is equivalent to the time taken from point A (i.e., the buffer head of the upstream node) to point D (i.e., the buffer head of current link channel l). Specifically, this time consists of two parts. The first part represents the delay to traverse the router datapath and the link. For WH router, this traversal delay is a constant value which equals to the number of pipeline stages in the router and the link (i.e., H in Table II). The second part accounts for the waiting time of a flit stalling at the buffers before arriving at point D. This is because if the downstream channels of link l are congested, the backpressure will stall the flit transmission even if this flit has been granted the use of the link. The flit may be stalled at any buffer slot between points A and D. Therefore, the second part delay is approximated by the waiting time of a queuing system with capacity $B+1$ (B is the buffer size of input port while 1 accounts for the buffer slot holding the flit in the upstream link, i.e., buffer slot of point A). To solve this queuing system, both arrival and service processes should be characterized. As shown in Fig. 3(a), the mean flit arrival rate λ_l^{flit} of this queuing system can be obtained by aggregating the flits sending toward the link l [3], [8]: $\lambda_l^{\text{flit}} = L \times \sum_{f \in F_l} \lambda_f$.

In order to characterize the service process, let s_l^{flit} represent the mean time to serve a flit in the queuing system of Fig. 3(b). Then s_l^{flit} can be calculated as the weighted average of the service time for every flow f passing through the link l [11]. More specifically, from Table II and Fig. 3(a), with $d(f, l, 1)$ denoting the next link of l in the routing path of f, s_l^{flit} can be calculated as

$$s_l^{\text{flit}} = \frac{\sum_{\forall f \in F_l} \left[\lambda_f \times \left(\frac{h_{d(f,l,1)}}{L} + \frac{1}{1 - Pb_{d(f,l,1)}} \right) \right]}{\sum_{\forall f \in F_l} \lambda_f}. \quad (2)$$

Equation (2) shows that for a packet with L flits, it first takes $h_{d(f,l,1)}$ cycles for its header flit in the buffer to win the access of the next link (i.e., link $d(f, l, 1)$). After that, all the L flits in the packet are sent to the link $d(f, l, 1)$ directly without further arbitration. If we denote the blocking (i.e., the buffer full) probability of link $d(f, l, 1)$ as $Pb_{d(f,l,1)}$, which equals to the probability that there are $B+1$ customers in the queue and can be obtained from solving the queuing system of link $d(f, l, 1)$, then the average time to transfer a single flit toward link $d(f, l, 1)$ can be approximated as $1/(1-Pb_{d(f,l,1)})$ similar to the derivations in [5]. Overall, the mean service time at link l for flits in flow f is obtained by summing these two parts together, which equals to $h_{d(f,l,1)}/L + 1/(1 - Pb_{d(f,l,1)})$.

After the mean customer arrival rate (i.e., λ_l^{flit}) and the average service time (i.e., s_l^{flit}) of this queuing system are obtained, the queuing formulations presented in [5], [11], and [27] can be used to obtain the estimation of the queuing time, as well as the link blocking probability Pb_l. Of note, Pb_l will be used by its upstream links as in (2). Finally, the flit transfer time over link l (i.e., η_l) is obtained by adding this delay with the router and link traversal time H.

2) Path Contention Time Calculation: The path contention delay h_l for link l is defined as the average allocation time for the packet to be granted the link l after contention with other flows forwarding toward the same link; it is usually modeled as the waiting time of a queuing system shown in Fig. 3(c) [11], [14]. For fair allocation policies such as round-robin arbitration, each flow has the same priority and takes turns to use the output link. Therefore, the system capacity K of the equivalent queue in deriving h_l is approximated as the number of flows that contend for the current channel [11], [14]. For example, in Fig. 3, $K = 3$ for the link l.

The arrival process to the queuing system in Fig. 3(c) can be obtained by merging all flows that route to l. Therefore, the mean packet arrival rate is [6]: $\lambda_l = \sum_{f \in F_l} \lambda_f$. According to the GE traffic model, for a specific flow f passing through

link l, the probability that the packets of f are generated following the exponential branch is $2/(1+C_f^2)$ [10], [20]. For a link l with multiple flows, in the GE traffic model, the overall probability of packets going through the exponential branch equals to the sum of each individual probability of flow f passing through the link [20]. Therefore, the SCV of the aggregated traffic on link l (i.e., C_l^2 in Table II) need to satisfy the following equation [20]: $2/(1+C_l^2) = \sum_{\forall f \in F_l}(\lambda_f \times 2/(C_f^2+1))/\sum_{f\in F_l}\lambda_f$.

The service process of the queuing system for computing h_l is also illustrated in Fig. 3(a) and (c). Here, we adopt the method proposed in [11] to calculate the service time which is the time interval that a packet takes to traverse the link l. In Fig. 3(a), assume a header flit in point A is granted the access of the current link l. Then, the service process of this packet begins when the header flit leaves A and ends when the tail flit arrives at the same point to release the channel [11]. If the downstream links are not congested, this service time simply equals the packet length (i.e., L cycles) because the entire packet can traverse the link continuously [11]. However, when there is a severe blockage along the path, the worst case transfer scenario requires the packet head reaches point B [Fig. 3(a)] where the overall buffer spaces from points C to B are L flits (i.e., the entire packet size) [11], [20], [23]. Formally, for link l, let Λ_l^f represent the effective number of downstream links that a packet of flow f crosses; Λ_l^f then equals to the smaller between L/B and the remaining hops from current link to the flow destination [11], [23]. Let x_l^f denote the time delay to move a flit from points A to B; x_l^f is then given by considering the contention and transfer delays along the next Λ_l^f links altogether [11], [20][1]

$$x_l^f = \eta_l + h_{d\left(f,l,\Lambda_l^f\right)} + \sum_{j=1}^{\Lambda_l^f-1}\left(\eta_{d(f,l,j)} + h_{d(f,l,j)}\right). \quad (3)$$

Different from [11], [20], and [23], we have included the router and link pipeline delay H in the calculation of $\eta_{d(f,l,j)}$, which therefore incorporates the traversal delays of downstream routers when approximating the worst-case bound x_l^f. The service time s_l^f for flow f is then bounded by the packet length L (lower bound) and x_l^f (upper bound), which is approximated as [11], [20], [23]

$$s_l^f = \begin{cases} \left[L\times\left(L+x_l^f\right)+2\times x_l^f\times L\right]\Big/\left(L+2x_l^f\right) & \text{if } x_l^f < L \\ \left[L\times\left(L+x_l^f\right)+2\times\left(x_l^f\right)^2\right]\Big/\left(L+2x_l^f\right) & \text{otherwise.} \end{cases} \quad (4)$$

Equations (3) and (4) show that if the flit transfer time η and the packet allocation delay h of the downstream links in flow f are known, the average service time s_l^f with respect to flow f can be obtained. The overall service time of the queuing system in Fig. 3(c) can be computed by averaging over all flows f passing through the link l, which produces the mean service time s_l as [11]: $s_l = \sum_{\forall f\in F_l}(\lambda_f \times s_l^f)/\sum_{\forall f\in F_l}\lambda_f$.

[1] z_l in Fig. 3(a) is the additional time that the packet reaches to the exact buffer slot holding the whole packet and is usually neglected in x_l [11], [23].

The SCV of the service process for link l can be approximated in a similar manner as discussed in [6]

$$R_l^2 = \frac{\overline{\left(s_l^f\right)^2}}{(s_l)^2} - 1 = \left(\frac{\sum_{\forall f\in F_l}\lambda_f \times \left(s_l^f\right)^2}{\sum_{\forall f\in F_l}\lambda_f}\right)\Big/(s_l)^2 - 1. \quad (5)$$

After characterizing the arrival and the service processes of the queuing system in Fig. 3(c), we then use $M/G/1/K$ queuing formula [5], [11], [27] to obtain the waiting time h_l^*. Next, the path contention delay h_l is calculated by considering the second moment (SCVs) of the traffic input. In this paper, h_l is approximated as the product of h_l^* and the waiting time ratio between $G/G/1$ and $M/G/1$ queuing systems

$$h_l = \frac{\left(R_l^2 + C_l^2\right)}{\left(1+R_l^2\right)} \times h_l^* \quad (6)$$

where R_l^2 and C_l^2 are the SCVs of the service and interarrival times of link l, respectively. $(R_l^2+C_l^2)/(1+R_l^2)$ is the ratio derived from the general $G/G/1$ and $M/G/1$ queuing formula in [6].

3) Source Waiting Time Calculation: The packet injection queue at the network interface is usually modeled as a queuing system with an infinite capacity [14]. If we denote the link connecting the source PE s and the attached router as l, then the waiting time v_s can be calculated based on a $GE/G/1/\infty$ queuing model as [10]

$$v_s = \frac{s_l}{2}\left(1 + \frac{C_l^2 + \lambda_l \times \frac{(s_l-L)^2}{s_l}}{1-\lambda_l \times s_l}\right) - s_l. \quad (7)$$

In (7), the first two moments of the input traffic characteristics (i.e., λ_l, C_l^2), as well as the mean service time s_l are derived following similar procedures in normal link channels.

IV. New SVR-NoC Latency Model

To further improve the prediction accuracy of the proposed queuing model, in this section, we present the SVR-NoC latency model to refine the queuing predictions.

A. Channel and Source Queuing Regression Models

Based on (1), we define two delay metrics in the learning model. The first is the source queuing delay v_s. The second is the channel waiting time w_l, which includes the contention and transfer delay for a flit to use link l. Specifically, w_l equals to $h_l + \eta_l$ in the proposed analytical model. In this paper, we model these two components via two regression functions f_{SQ} (i.e., the source queuing delay function) and f_{CQ} (i.e., the channel queuing delay function). We denote the features used in the learning by two vectors: $X_{\mathrm{CQ}} = [x_{\mathrm{cq1}}, x_{\mathrm{cq2}}, \ldots, x_{\mathrm{cqn}}]$ and $X_{\mathrm{SQ}} = [x_{\mathrm{sq1}}, x_{\mathrm{sq2}}, \ldots, x_{\mathrm{sqn}}]$. Then, the predictions of these two metrics can be obtained as [19]

$$w_l = f_{\mathrm{CQ}}(X_{\mathrm{CQ}}); \qquad v_s = f_{\mathrm{SQ}}(X_{\mathrm{SQ}}). \quad (8)$$

Using (8), given the features of any new traffic input, we can estimate v_s and w_l at the source and link channels, respectively. Substituting them in (1) gives the latency prediction of each specific flow. Of note, for an NoC topology other than meshes,

TABLE III
TRAINING FEATURES FOR REGRESSION

Feature vector	Elements notation	Elements	Description
X_{CQ}	λ_l	$X_{CQ}[1]$	The packet arrival rate at link l
	Λ	$X_{CQ}[2:5]$	The vector whose k^{th} entry represents the aggregated traffic from other input channels and contends with link l towards the output direction k
	F	$X_{CQ}[6:9]$	The forwarding probability vector whose j^{th} element represents the routing ratio from current channel to the output direction j
	w_l	$X_{CQ}[10]$	The analytical channel queuing time $\eta_l + h_l$
	s_l	$X_{CQ}[11]$	The analytical channel service time of the link l
X_{SQ}	λ_s	$X_{SQ}[1]$	Packet arrival rate at the source PE s
	F	$X_{SQ}[2:5]$	The forwarding probability vector whose j^{th} element denote the ratio of traffic from PE s to the output direction j
	Λ	$X_{SQ}[6:9]$	The vector whose k^{th} entry denotes the aggregated traffic that contends with local PE s for the downstream link k
	W	$X_{SQ}[10:13]$	The vector whose j^{th} entry represents the analytical waiting time of the downstream link j (*i.e.*, $h_j + \eta_j$)
	S	$X_{SQ}[14:17]$	The vector whose j^{th} element denotes the analytical service time of the downstream link j (*i.e.*, s_j)
	v_s	$X_{SQ}[18]$	The analytical source queuing delay
	s_l	$X_{SQ}[19]$	The analytical service time of the injection link l

we can still characterize that topology as a LDG. After collecting training data on the corresponding new topology, the channel and source waiting time models can be learned and used in a way similar to (8).

B. Feature Extraction From the Training Data

1) Channel Queuing Feature Vector: The average waiting time in the link l (i.e., w_l) depends not only on its packet arrival rate but also on the contentions among multiple channels in the same router, as well as the traffic conditions in the neighboring routers. In the analytical models, these effects are usually calculated explicitly by computing the contention matrix among the channels and the forwarding probability matrix inside each router [3], [5]. In contrast, in SVR-NoC, we aim at learning the impact of channel contentions implicitly by providing the SVR engine with sufficient samples. Toward this end, the channel queuing feature vector X_{CQ} is made up of three parts.

1) We first include the traffic arrival rate λ_l of link l to capture the traffic workload injected to the channel.
2) The forwarding probability vector F of the current channel to different output directions as well as the amount of traffic Λ from other input channels going to the same output direction are included to reflect the contentions from other input ports.
3) Finally, the analytical link service time s_l and waiting time $\eta_l + h_l$ obtained from the proposed queuing model are included to provide an analytical estimation for w_l.

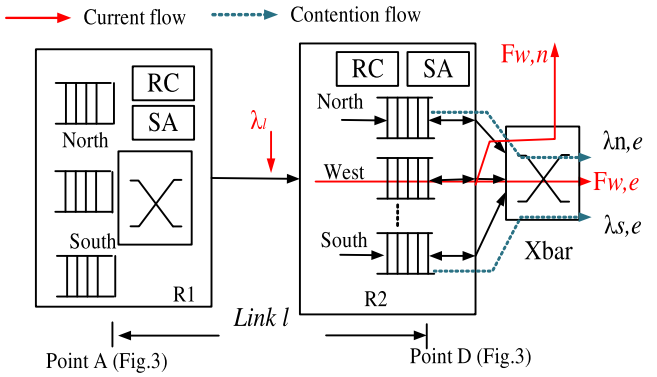

Fig. 4. Illustration of X_{CQ}: for a link channel connecting routers R1 and R2, λ_l is the aggregated traffic. $F_{w,e}$ and $F_{w,n}$ represent the probability of routing toward the east and north output ports, respectively. Considering contentions in R2, $\lambda_{n,e}$ and $\lambda_{s,e}$ represent the amount of traffic that comes from the north and south input of R2 and competes with the west input for the east output link.

Of note, both arrival rates and forwarding probability vector F are defined in [3] and can be calculated offline.

In Table III, we summarize these three elements in the channel queuing feature vector X_{CQ}. The features are further illustrated in Fig. 4. More specifically, as shown in Fig. 4, for the current west input channel, when considering the east output direction, the forward probability $F_{w,e}$ represents the portion of traffic that will be forwarded toward east output. Formally, $F_{w,e} = \lambda_{w,e}/\lambda_w$, where $\lambda_{w,e}$ is the amount of traffic traversing from the west port toward east output direction; λ_w is the aggregated traffic at the west input port [3]. Moreover, flows from other input channels routing toward the east output port also need to be considered. In Fig. 4, $\lambda_{s,e}$ and $\lambda_{n,e}$ indicate the traffic from the south and north input channel that contends with west input port for the same east output direction, respectively. Therefore, in this example, the input feature vector X_{CQ} contains an entry of $(\lambda_{s,e} + \lambda_{n,e})$ which records the aggregated traffic contending with current channel for the east output port.

Compared to the learning model proposed in [19], two modifications have been made in this paper. First, for the feature elements that are grouped in an array form such as $\Lambda = X_{\mathrm{CQ}}[2{:}5]$ in Table III, each element corresponds to a different output direction (excluding the input channel itself). In this paper, the values in the array (e.g., $X_{\mathrm{CQ}}[2] - X_{\mathrm{CQ}}[5]$) are rearranged in descending order instead of the physical direction order. For example, for the west (W) input channel in Fig. 4, $X_{\mathrm{CQ}}[2{:}5]$ records the aggregated traffic that contends with input channel for the north (N), south (S), east (E), and local (L) output directions, respectively. Previously, in [19] $X_{\mathrm{CQ}}[2{:}5]$ are filled following the "*N-E-S-W-L*" order and the direction of the channel itself (i.e., W) is automatically skipped. However, we noticed that in the SVR-learning, when making predictions, $X_{\mathrm{CQ}}[2{:}5]$ are compared element-by-element with the support vectors to determine the level of similarity. Because flow contentions are independent of the

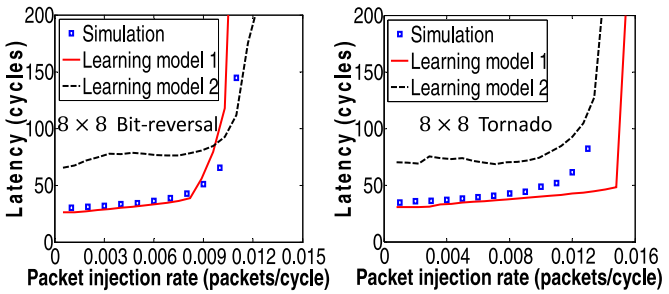

Fig. 5. Comparisons of the learning accuracy for the models with and without analytical queueing features under bit-reversal and tornado traffic patterns. The learning model 1 is the model which includes the analytical predictions from the proposed queuing approach. The learning model 2 refers to the model without using features of queuing estimations.

physical output direction, a more reasonable organization is to arrange the values of $X_{\mathrm{CQ}}[2{:}5]$ in descending order. This way, for example, $X_{\mathrm{CQ}}[2]$ has a new meaning which indicates the largest amount of traffic that contends with the current channel for a certain output direction.

Second, for the X_{CQ} vector, the analytical channel queuing time is also included in X_{CQ}. This is motivated by the observation in Fig. 5, where we compare the regression accuracy between two learning models. In Fig. 5, all the features in the learning model 2 are collected from simulations similar to that in [19] without using the inferences from the proposed analytical model. Two limitations exist for some traffic patterns according to our experiments.

1) For some untrained traffic patterns, such as bit-reversal, the model accuracy degrades as there are no similar support vectors provided in the training stage.
2) Since the delay values w_l (as well as v_s) have a large dynamic range, in order to minimize the distance among all the training data, the learning model puts more effort on minimizing the mean square error (MSE) in the heavy workload region. As a result, the accuracy under light and zero workload is sacrificed as shown in Fig. 5.

On the other hand, if we include the estimations obtained from the proposed analytical model (i.e., the learning model 1 shown in Fig. 5), the regression precision can be further improved.

2) Source Queuing Feature Vector: For the source queuing feature vector, the packet generation rate λ_s is first included in X_{SQ} as shown in Table III. Moreover, to achieve a better inference on the network congestion level, we also include the analytical waiting times of the downstream links in the X_{SQ} vector (i.e., W vector in Table III). Specifically, in the W vector, its jth element w_j represents the queuing-theory-based waiting time of the downstream link j in the attached router of the PE s. In this way, a higher w_j value indicates the traffic from PE s may have a higher chance to be blocked if routing toward that direction. Finally, similar to the channel queuing feature set, we include the calculated source queuing time of v_s in the feature set to improve the learning performance.

C. Calculating Analytical Features in SVR-NoC

In SVR-NoC, the proposed analytical model is embedded into the learning process to form the feature vectors.

Algorithm 1 Flow to Obtain Queuing Predictions of w_l and v_s

Input: F the application flow set; G the ordered list of links for queuing analysis
Output: w_l the estimated channel waiting time for link l; v_s the analytical source waiting time of node s
for all link $l \in G$ **do**
  (λ_l, C_l^2) = traffic_model(F, l)
  $\eta_l = calculate\_transfer\_time(\lambda_l^{flit}, s_l^{flit}, L, B)$;
  for all $f \in F_l$ **do**
    $s_l^f = calculate\_link\_service\_time()$;
  end for
  $(s_l, R_l^2) = aggregated\_service\_time()$;
  if $from\_node(l) \neq to\_node(l)$ **then**
    $k = number\_contention\_flow(l)$;
    $h_l = calculate\_contention\_delay(\lambda_l, C_l^2, s_l, R_l^2, k)$;
    $w_l = h_l + \eta_l$;
  else
    $v_s = calculate\_source\_queuing(\lambda_l, C_l^2, s_l, R_l^2)$;
  end if
end for

The steps to obtain the queuing estimations are summarized in Algorithm 1 [13]. For the input application, we first apply the link dependency analysis presented in Section III-B to determine the correct link ordering. Then, for each link l in the ordered list, we calculate the arrival traffic model (λ_l, C_l^2) according to the routing algorithm and applications. As discussed in Section III and [13], the link transfer time η_l only depends on its downstream link contention time [i.e., $h_{d(f,l,1)}$ in (2)], which should have already been obtained during the previous iteration. On the other hand, the path contention delay h_l depends not only on the contention delay and transfer time of its downstream links but also on the current link's η_l [(3) and (4)]. Therefore, η_l is calculated first. After η_l is obtained, we then calculate the mean and SCV of the link service time (s_l, R_l^2). After the queuing system being characterized, (6) and (7) are used to predict h_l and v_s, respectively. Finally, the (h, v, η) queuing variables are used to form X_{CQ} and X_{SQ} vectors based on Table III.

Fig. 6 summarizes the overall working flow of SVR-NoC, which consists of two parts, namely the training stage and prediction stage [19]. In both stages, the links are first ordered based on the LDG extracted from the input traffic. Then, Algorithm 1 is applied to obtain the queuing estimations of w_l and v_s (named as wq and vq in Fig. 6) for each link. Of note, during the training stage, the input traffic patterns are synthetically created (e.g., the random and transpose traffic patterns). The collected simulation results then form a training data set for regression. After the training stage, the obtained SVR models are used in the prediction tool to evaluate the latency performance for the new applications. Specifically, the features in X_{CQ} and X_{SQ} are computed based on the new input and the proposed queuing model. Then, the f_{CQ} and f_{SQ} functions are used to evaluate the channel and source waiting times (i.e., w_l and v_s) based on (8). Finally, by aggregating the links in the routing path of a flow based on (1), the end-to-end latency can be obtained.

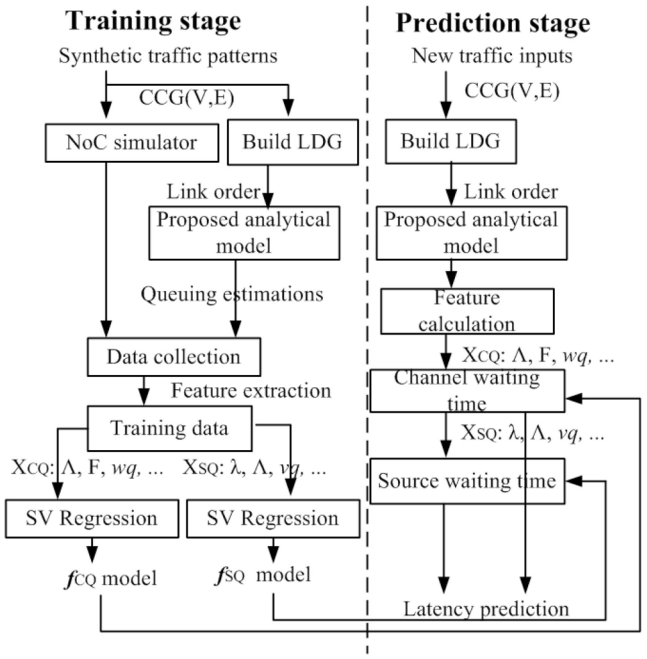

Fig. 6. SVR-NoC working flow: the input is the application CCG(V, E) [3], [6], where V represents the set of cores and E represents the set of communication flows in the application. Λ and F represent the arrival rate matrix and forwarding probability matrix described in Table III, respectively. wq and vq correspond to the channel waiting and source queueing time predicted by the analytical model.

D. Support Vector Regression for f_{SQ}

After obtaining the feature vectors X_{CQ} and X_{SQ}, we apply the standard ϵ-SVR [17] techniques to learn the two nonlinear models f_{CQ} and f_{SQ}, respectively. Summarized in [16], [17], [29], and [30] the ϵ-SVR formulates a primal problem first; then the problem is solved in the dual space; finally, the model is extended using kernel functions. In the following, we explain the ϵ-SVR formulation of the source queuing function f_{SQ}, while similar procedures are also applied for f_{CQ}.

1) Preprocessing the Source Queuing Feature and Label: The source queuing time usually has a very large dynamic range which ranges from nearly zero (i.e., light traffic load) to hundreds or even thousands of clock cycles (i.e., heavy traffic load). The extremely large delay usually indicates a near-saturation or unstable network condition. Without scaling, the SVR will unnecessarily put efforts on fitting the data in these regions which sacrifices the model accuracy at light and medium traffic loads. Because of this, in SVR-NoC, we preprocess the simulated v_s label and the analytical v_s features using a simple function $f(x) = x/(x + c)$ (where c is empirically chosen to be ten according to our experiments; x is the v_s variable from the simulation or analytical model). As shown in Fig. 7(a), after preprocessing, all the delay values fall into the region $[0, 1)$. In addition, the extreme large delay values will not cost additional effort for following procedures.

2) Primal Problem Formulation: Without loss of generality, SVR begins by assuming f_{SQ} to be a linear function of X_{SQ} [17]. Specifically, assume the vector X_{SQ} has d features, then under the linear model assumption, v_s can be expressed as [16], [29]: $v_s = f(X_{SQ}) = \sum_{i=1}^{d} w_i x_{sqi} + b = \mathbf{w}^{\mathbf{T}} X_{SQ} + b$, $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$, where w_i and b represent the weights of the ith feature and the bias, respectively.

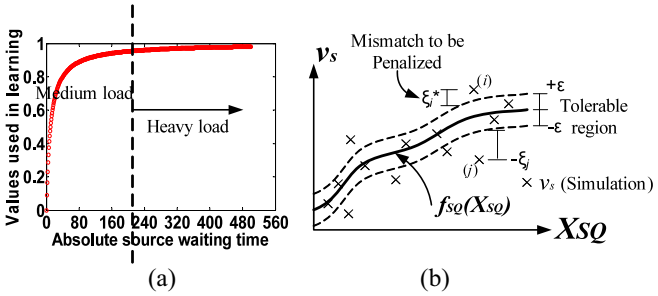

Fig. 7. (a) Preprocessing function for the source queuing delay. The input variable x is the source waiting time v_s. The function output is used instead of v_s in the learning. (b) Illustration of the ϵ-SVR technique [16], [17], [29], [30]: the objective is to learn a function $f_{SQ}(X_{SQ})$ which tolerates the mismatch between the function output and the simulation points by $[-\epsilon, \epsilon]$; the additional derivation (e.g., ξ_i^* and ξ_j with respect to point i and j in the figure) will be penalized [29].

When learning the source queuing function, at the training stage, n data samples, i.e., $\{(X_{SQ1}, v_{s1}), \ldots, (X_{SQn}, v_{sn})\}$, are collected from simulations. As shown in Fig. 7(b), ϵ-SVR aims to minimize the differences between the regression function and the n training data points [16], [17]. More specifically, ϵ is defined as the maximum tolerable difference between the function and the data point [29]. For any training data sample, the amount of fitting error which is larger than the $[-\epsilon, \epsilon]$ region needs to be penalized as shown in Fig. 7(b) [16], [29]. In order to prevent data over-fitting when learning $f(X_{SQ})$, a regularization term proportional to $||\mathbf{w}||^2$ is added into the objective function [16], [17]. In summary, the problem is formulated in [16], [17], and [29]

$$\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{j=1}^{n}\left(\xi_j + \xi_j^*\right) \tag{9}$$

$$\text{s.t.} \quad \begin{cases} \left(\mathbf{w}^T X_{SQj} + b\right) - v_{sj} \le \epsilon + \xi_j, \forall j \in [1, n] \\ v_{sj} - \left(\mathbf{w}^T X_{SQj} + b\right) \le \epsilon + \xi_j^*, \forall j \in [1, n] \\ \xi_j, \xi_j^* \ge 0, j \in [1, n], \epsilon \ge 0 \end{cases} \tag{10}$$

where the parameter C is used to explore between improving the fitting precision [the second term in (9)] and reducing the chance of over-learning [the first term in (9)].

3) Solving Dual Problem and Model Extension: To solve the primal form optimization problem in (9) and (10), a set of Lagrangian multipliers α_j, α_j^* are introduced and the corresponding Lagrangian dual function $\mathcal{L}$ is derived as in [16], [17], and [29]. The optimal solution of the dual problem is obtained by solving the equations where the partial derivatives of $\mathcal{L}$ on variables $(\mathbf{w}, b, \xi_j, \xi_j^*)$ equal to zero [16], [29]. Following the derivations in [16], [17], and [29], the dual problem can be solved efficiently with the following form [29]:

$$f_{SQ}(X_{SQ}) = \sum_{j=1}^{n}\left(\alpha_j - \alpha_j^*\right) < X_{SQj}, X_{SQ} > + b. \tag{11}$$

In [16] and [29], it has been proven the solution in (11) can be extended to nonlinear models by using the kernel function k instead of the original inner product [16], [30]

$$k(X_{SQ}, X_{SQj}) = < \Phi(X_{SQ}), \Phi(X_{SQj}) > \tag{12}$$

where $< \cdot , \cdot >$ in (11) and (12) represents the dot (inner) product of two vectors [29]. From the analytical delay model, it is suggested that the router delay is a nonlinear function of the extracted features. Therefore, in this paper, the widely adopted radial basis function (RBF) is used as the kernel [16], [17], [30]: $k(X_{\mathrm{SQ}}, X_{\mathrm{SQ}j}) = \exp(-\gamma \cdot ||X_{\mathrm{SQ}} - X_{\mathrm{SQ}j}||^2)$, where γ is a tuning hyper-parameter. As using the RBF kernel in (11), the f_{SQ} function is extended to a nonlinear form with the feature inputs.

4) Exploration of Hyper-Parameters: In ϵ-SVR, there are three hyper-parameters, i.e., ϵ, C and γ, need to be determined [17], [29], [30]. In this paper, we adopt a v-fold cross-validation approach to explore the optimal hyper-parameter and avoid data over-fit [16], [17]. In general, the v-fold cross-validation separates the training data into v sub-groups [16]. For each hyper-parameter combination, v iterations are needed; every time, $v-1$ subsets are used in the regression and the remaining subset is for validation [16]. The final hyper-parameters are chosen corresponding to the minimal MSE across the v sets [16]. In this paper, we explore the parameter settings on ϵ, C, and γ using a tenfold cross-validation. The range and resolution of the hyper-parameters are chosen according to examples in [31] and [32].

V. Experimental Results

A. Experimental Setup and Training Data Preparation

We use Booksim2.0 [33] to evaluate the NoC performance on mesh NoCs. For all comparisons, we assume a two cycle pipelined router architecture. The link traversal cycle is assumed to be one. Therefore, for a packet consisting of L flits and traversing N hops, the zero load latency L_{zero} can be estimated as $L_{\mathrm{zero}} = (2+1) \times N + (L-1) = 3N + L - 1$. For each mesh size (ranging from 4×4 to 12×12), various synthetic traffic patterns including uniform random, transpose, shuffle, and tornado [21] are used as inputs to the target router architecture to obtain the training dataset. The training traffic are generated based on the Poisson traffic model. To verify the latency performance, after the training stage, the learned model is used to predict the delay with different injection rates for these patterns. We also use GE-traffic model discussed in Section III-C to generate the bursty traffic patterns for evaluation. The SCV is used to control the levels of burstiness. Moreover, we use the obtained learning model to predict the delay for other traffic patterns that are different from the training sets. Here, we use bit-reversal and bit-complement [21] traffics.

The SVR-NoC is implemented in MATLAB using functions from Libsvm and LS-SVM libraries [31], [32]. Besides using synthetic traffic, real-application traffic patterns are also used. For multi-processor system-on-chip (MPSoC) benchmarks, the multimedia system (MMS) [34], MPEG4 (MPEG4 codec) [35], video object plane decoder (VOPD) [35], multi-window decoder (MWD) [35], and two E3S applications (auto-indust and consumer) [36] are used. For chip multi-processor (CMP) applications, traces extracted from SPEC applications [37] are used. Specifically, the traces include IBM, Oracle, and Apache server applications, the scientific computation and the ocean simulation applications [37]. In the experiments, each time, we choose four applications and mapped them on an 8×8 mesh NoC platform to model the application consolidation scenario as in [38]. The generated traces are named as mix-1–mix-5 in the experiments.

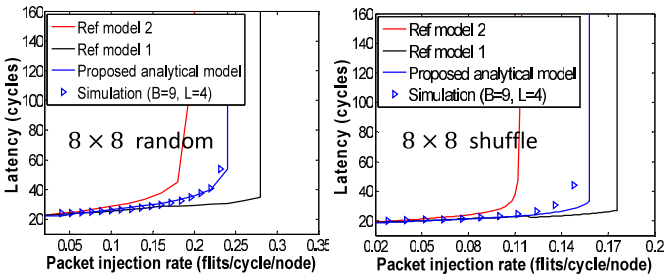

Fig. 8. Analytical model accuracy comparisons: the simulation results (blue triangles) are compared against the proposed analytical model (blue line), the reference model 1 from [6], and the reference model 2 from [3].

B. Proposed Queuing Model Accuracy

We first evaluate the proposed analytical model using two synthetic traffic patterns (i.e., random and shuffle traffic) with Poisson packet injection rates at each source node. The packet length L is assumed to be four flits. The buffer depth B is nine flits per input port. We compare the proposed model with two representative NoC analytical models. The reference model 1 is adopted from [6], where the priority $G/G/1$ queue has been modified to a generalized $G/G/1$ queue formula. The reference model 2 is based on the $M/G/1$ queuing model proposed in [3]. In general, the accuracy of analytical models depend on the traffic patterns and network sizes. For larger mesh network and uneven traffic patterns, since the contentions at each link become more complex, the analytical models usually have a larger prediction error. In Fig. 8, we show a typical comparison of these three analytical models under an 8×8 mesh size.

As shown in Fig. 8, the two reference models either over-estimate or under-estimate the network saturation injection points by 17.3%–25.3%, while the proposed model achieves less than 6.7% error in predicting the network saturation point [39] for these two traffic patterns.

C. Prediction Accuracy of the SVR-NoC Learning Model

1) SVR Learning Performance: In Table IV, we summarized the learning performance for the f_{SQ} function, while similar results can be observed for the channel queuing function. Three types of datasets are compared. The Dataset-1 uses four synthetic traffic patterns for training, i.e., random, transpose, shuffle, and tornado, which contain 23 760 training samples. The bit-reversal and bit-complement traffic patterns are used for testing, which contain 9744 samples. In the Dataset-2, the tornado traffic pattern has been moved from the training set to the test set. As a result, Dataset-2 contains 17 616 samples for the training and 15 888 samples for the testing. For the Dataset-3, we only used random traffic in the training and all other data have been moved to the testing data set. In Table IV, we also evaluate the sensitivity of features on the learning performance. Specifically, Feature set 1 includes all the features explained in Section IV-B [i.e., $X_{\mathrm{SQ}}(1{:}19)$ in Table III]. Feature set 2 excludes a single analytical feature, i.e., the source queuing time $X_{\mathrm{SQ}}(18)$ from Feature set 1.

TABLE IV
NRMS ERROR AND SQUARED CORRELATION COEFFICIENT (C2) COMPARISONS FOR f_{SQ} FUNCTION

Datasets		Dataset-1						Dataset-2						Dataset-3					
Model		SVR		Poly		Lin		SVR		Poly		Lin		SVR		Poly		Lin	
Metrics		C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS	C2	NRMS
Feature-1	train	0.9043	0.3265	0.9532	0.2270	0.8462	0.4138	0.8990	0.3363	0.9365	0.2682	0.8115	0.4740	0.7972	0.4618	0.8250	0.4258	0.7150	0.5551
	test	0.8704	0.3789	0.7288	0.5209	0.7784	0.5150	0.8682	0.3825	0.6889	0.5611	0.8039	0.5007	0.7938	0.5667	0.6781	0.6513	0.7890	0.6049
Feature-2	train	0.9036	0.3270	0.9691	0.1838	0.8473	0.4097	0.8968	0.3394	0.9518	0.2321	0.8064	0.4800	0.7975	0.4591	0.8388	0.4100	0.7110	0.5594
	test	0.8318	0.4308	0.1512	1.1048	0.7465	0.5794	0.8260	0.4347	0.1208	1.1528	0.8022	0.5345	0.6880	0.5907	0.3298	1.0028	0.8188	0.6702
Feature-3	train	0.9408	0.2623	0.9933	0.0886	0.9427	0.2625	0.9380	0.2660	0.9783	0.1595	0.8899	0.3532	0.9384	0.2664	0.9489	0.2481	0.8947	0.3665
	test	0.5737	0.6721	0.0070	1.7471	0.2661	0.9196	0.5070	0.8058	0.0074	1.4573	0.1802	0.9878	0.5598	0.7436	0.0092	1.5035	0.3951	0.7971

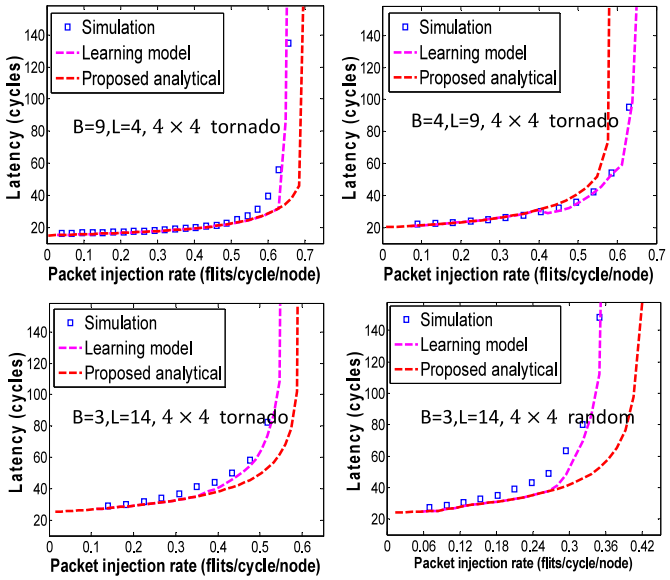

Fig. 9. Latency comparisons of the simulation results (square dots) against the proposed analytical model (red line) and the SVR-NoC learning model (purple line) under tornado and random traffic patterns for different buffer depth B and packet length L combinations.

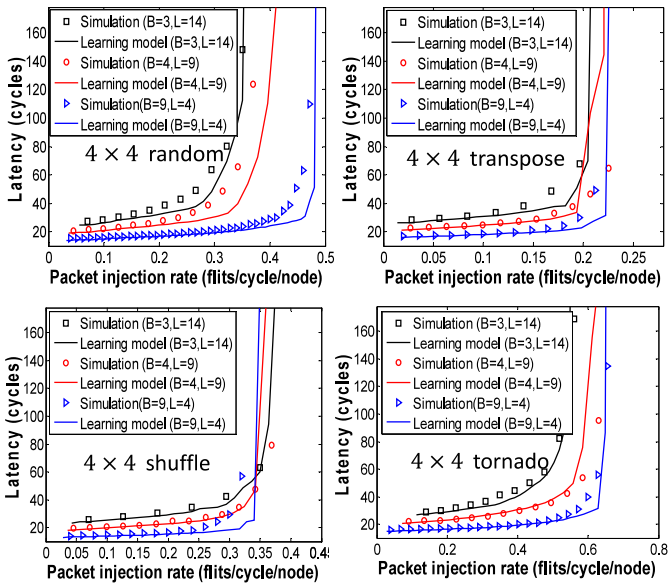

Fig. 10. Latency comparisons of the simulation and learning model under trained traffic patterns (random, transpose, shuffle, and tornado) on 4 × 4 meshes.

Feature set 3 excludes all analytical features. We have compared the SVR models against polynomial regression (Poly) and linear regression (Lin) for different dataset and feature set combinations. The widely adopted squared correlation coefficient (C2) and the normalized RMS [16] error (namely NRMS) metrics are used to evaluate the learning performance.

As shown in Table IV, by including more traffic patterns (e.g., from Dataset-3 to Dataset-1) into the training set, the learning model achieves better predictions (i.e., higher C2 and lower NRMS value) for the test traffic patterns. This is because different traffic patterns can provide wider coverage of the channel contention and forwarding features.

Moreover, comparing the learning results on Feature set 3 with Feature set 2 and 1, we can conclude the addition of analytical features significantly improve the learning accuracy. Feature set 1 uses one more analytical feature [i.e., $X_{SQ}(18)$] than Feature set 2 and further reduces the NRMS. Comparing the SVR learning with Poly and Lin regression models, it is also found the SVR with cross-validation achieves smaller error on the testing data.

2) Average Latency Prediction: In Fig. 8, it is shown the proposed queueing model provides a general estimation of different NoC configurations and improves the accuracy over the existing analytical models under random and shuffle traffic patterns. From our experimental results, it is still observed that the accuracy of the queueing model decreases under some traffic patterns due to the approximations and assumptions made in the derivation. To illustrate this, we first compare the SVR-NoC with the proposed analytical model. We consider different buffer depth B and packet length L combinations for a 4 × 4 mesh NoC architecture in the evaluation. The average latency obtained by SVR-NoC and the proposed analytical model are summarized in Fig. 9. From Fig. 9, it is shown that the proposed analytical model still incurs 8.9%–17.3% error in predicting the network criticality for some traffic patterns. For example, as shown in Fig. 9, for tornado traffic pattern, the error is 8.97%, 9.83%, and 13.5% for three different packet length (L flits) and buffer depth (B flits) combinations. The inaccuracy mainly comes from the simplifications made during the derivation of the queuing system such as the service time approximation in (4). On the other hand, when the SVR-NoC learning model is used to refine the queuing predictions, the accuracy can be further improved. As shown in Fig. 9, the learning model predicts the network saturation point accurately with less than 4.3% error for these two traffic patterns.

In Figs. 10–12, we show the prediction results for various traffic patterns and network configurations. The random, transpose, shuffle, and tornado traffic patterns are used in

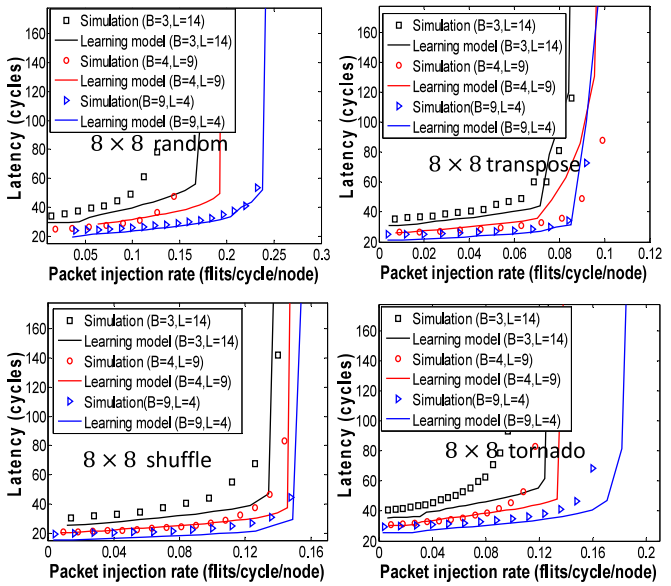

Fig. 11. Latency comparisons of the simulation and learning model under trained traffic patterns (random, transpose, shuffle, and tornado) on 8 × 8 meshes.

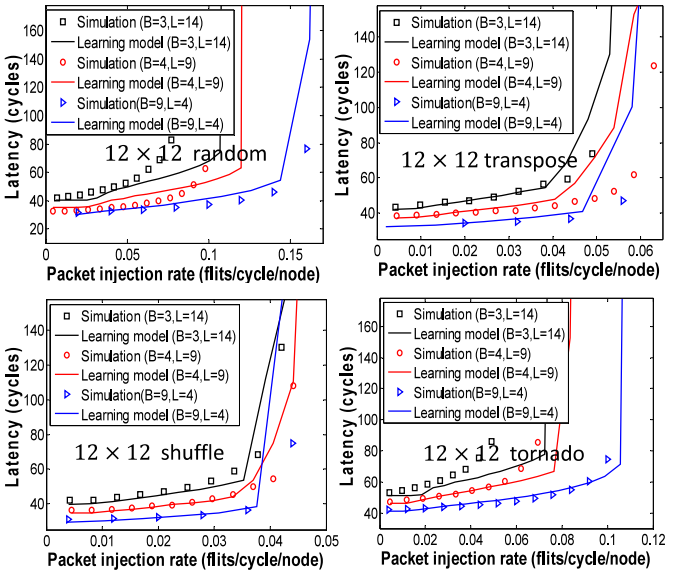

Fig. 12. Latency comparisons of the simulation and learning model under trained traffic patterns (random, transpose, shuffle, and tornado) on 12 × 12 meshes.

the training stage but with a different injection rate. Three router architectures are considered. They are WH routers with: 1) packet length $L = 4$ (flits) and buffer depth $B = 9$ (flits); 2) $L = 9$ and $B = 4$; and 3) $L = 14$ and $B = 3$. As can be seen from Figs. 10–12, for small mesh size (e.g., 4×4 mesh), SVR-NoC achieves the highest accuracy for both even and uneven traffic patterns. For a larger network, as a flow involves more and more links and each link introduces a certain error, the accuracy degrades. In general, we can still observe the SVR-NoC predicts the network saturation point accurately for most traffic patterns and NoC configurations.

3) SVR-NoC for Untrained Traffic and Real Applications: Next, we show the latency prediction performance of the SVR-NoC for the other traffic patterns that are not included in

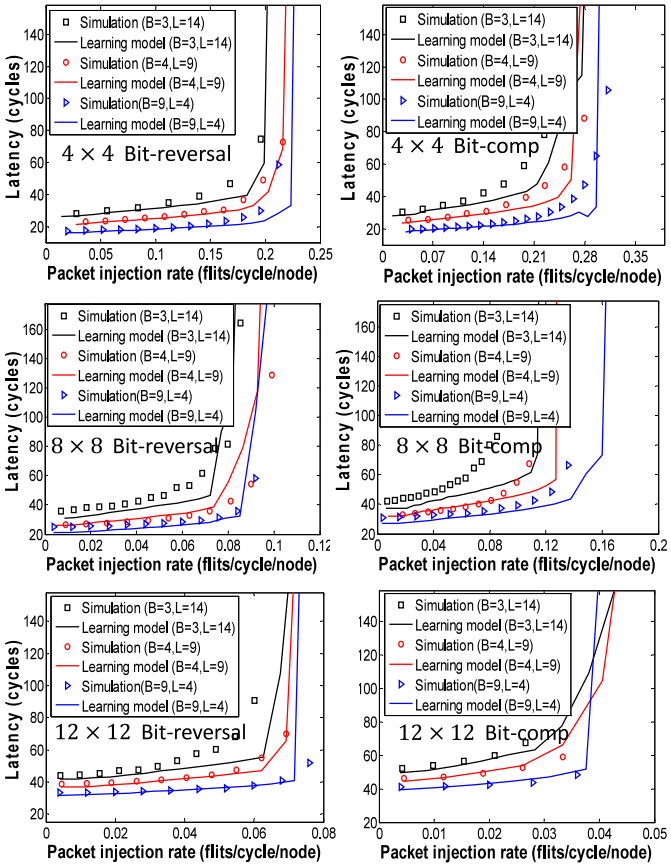

Fig. 13. Latency comparisons of the simulation and the learning model under untrained traffic patterns (bit-reversal and bit-complement) for 4 × 4, 8 × 8, and 12 × 12 mesh NoCs.

the training data set. Fig. 13 shows the comparison results for 4×4, 8×8, and 12×12 meshes. As shown in the figures, the SVR-NoC still demonstrates high accuracy in the bit-reversal and bit-complement traffic which are not included in the training set, with on average less than 12.5% error in predicting the network criticality status. In Fig. 13, a small bump is found for 4×4 the bit-complement traffic pattern. This is because the feature vectors X_{CQ} and X_{SQ} contain a bunch of features in addition to the injection rate. For this new traffic, some of the features (e.g., the forwarding matrices) may have a different range that is not seen in the learning, which causes a slightly different distances to the support vectors and introduces a fluctuation in the inference stage. One way to overcome such variation is to use some adaptive or batch-based learning techniques [16] which can dynamically include the traffic patterns with large errors to refine the learning model on-line.

We then evaluate SVR-NoC for synthetic traffic patterns with GE-type arrivals. Of note, in the training stage, the traffic provided to SVR-NoC are those injected following the Poisson process. Therefore, directly applying the learning model only yields the latency predictions for the case of SCV $= 1$. In order to model the latency under the traffic burstiness, which is characterized by SCV > 1, the predicted source and channel queuing times in (8) are further refined by the calculated SCV of the link according to (6) and (7). In Fig. 14, we show the SVR-NoC prediction results for two synthetic traffic patterns (transpose and bit-reversal) under various B and L combinations. Two bursty levels are considered

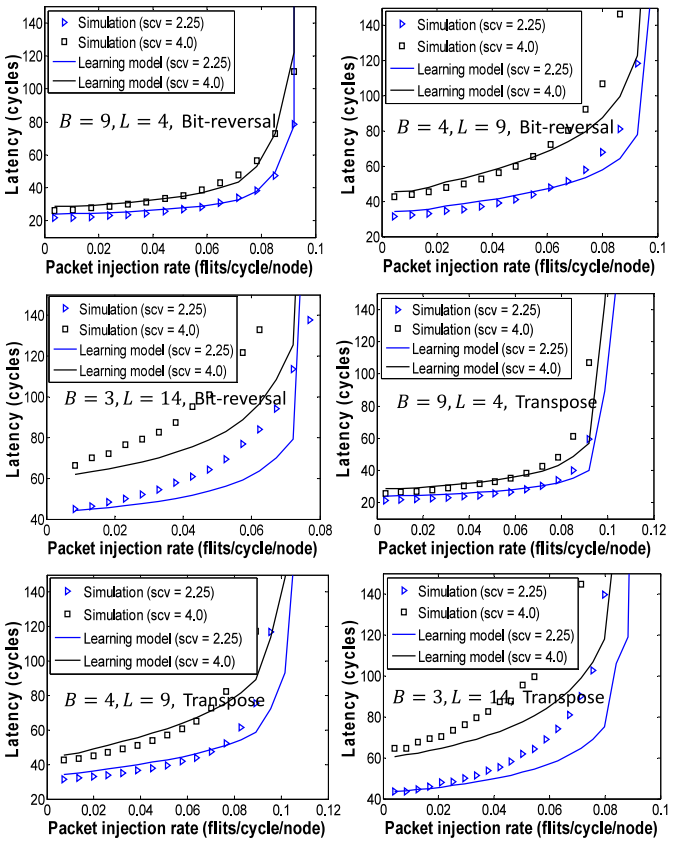

Fig. 14. Latency comparisons of the simulations and SVR-NoC model under GE-type traffic input on an 8×8 mesh NoC. Two traffic patterns are used in the evaluation, including the bit-reversal (un-trained) and transpose (trained) patterns. For each traffic pattern, three packet length L and buffer length B combinations are evaluated (i.e., $L = 9, B = 4$; $L = 4, B = 9$; and $L = 3, B = 14$). In each subfigure, two levels of burstiness are evaluated. The SCV equals to 2.25 and 4, respectively.

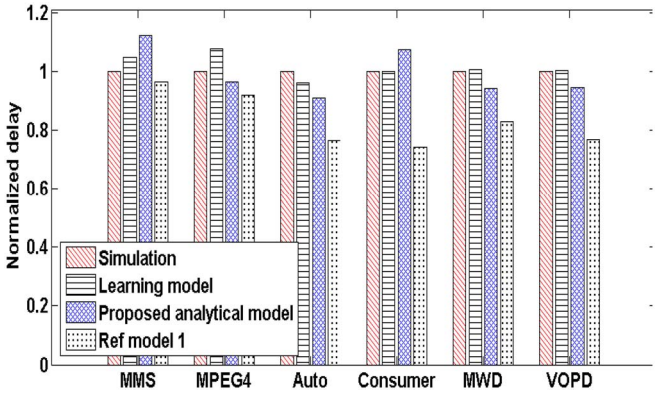

Fig. 15. Latency comparisons of various MPSoC benchmarks. The four bars correspond to the simulation, SVR-NoC learning model, proposed analytical model, and reference model 1 [6].

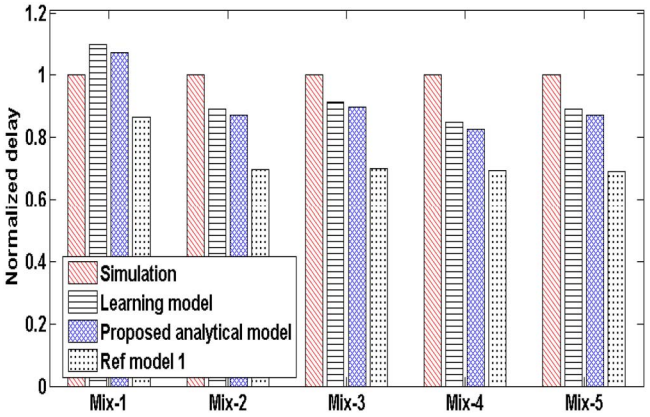

Fig. 16. Latency comparisons using the CMP applications. The four bars correspond to the simulation, SVR-NoC learning model, proposed analytical model, and reference model 1 [6], respectively.

(SCV = 2.25 and SCV = 4.0) which are chosen similar to the ranges used in [6]. As can be seen from the figure, the model prediction is worse for small B and large L values (e.g., $B = 3$, $L = 14$), this is because in this case, the packet crosses more more downstream channels. The analytical service time bounds in (3) and (4) introduce larger errors in this case than the other two B and L combinations. On average, the SVR-NoC achieves less than 12% error in predicting the network saturation points for these two traffic patterns under various configurations.

We also use several real MPSoC and CMP benchmarks to evaluate the prediction accuracy of SVR-NoC. For MPSoC benchmarks shown in Fig. 15, the traffic is injected according to the data rate characterized in the corresponding task graph. For the CMP traces shown in Fig. 16, the SCV of each flow f is calculated first before applying the analytical and learning models. We compare the learning model with the proposed analytical model and reference model 1 [6] which also uses SCV to reflect the levels of burstiness. Figs. 15 and 16 show the comparisons. As shown in figures, for two applications MPEG-4 and Mix-1, the learning model slightly over-estimates the latency compared to the proposed analytical model. This is because these two traffic patterns are not trained, sometimes the inferences based on the existing training data may over-compensate the queuing delay estimations. On average, for MPSoC applications, the SVR-NoC learning model achieves 3.0% error across the six traffic patterns; while the mean prediction errors for the proposed analytical model and reference model 1 are 7.2% and 16.8%, respectively. For CMP applications in Fig. 16, the average error of latency prediction for the learning model is 11.1%, while that for the proposed analytical and reference model 1 are 12.1% and 27.0%, respectively. In Fig. 17, we show the per-flow latency comparisons obtained from the simulation, the learning model, the proposed analytical model and reference model 1 [6] for the Consumer application in E3S benchmark [36]. As shown in Fig. 17, compared to the analytical models, the SVR-NoC learning model consistently achieves better accuracy.

D. Runtime Comparison

The training time of SVR-NoC for an 8×8 mesh with four synthetic patterns (i.e., Dataset-1 and feature set 1 in Table IV) takes about 1–2 hr.[2] However, the training process is taken off-line and it does not incur additional overhead during the performance prediction. The simulation time of an 8×8 mesh NoC for 1×10^6 cycles is about 10min while the SVR-NoC

[2]The time includes the cross-validation exploration for determining the hyper-parameters in the SVR model.

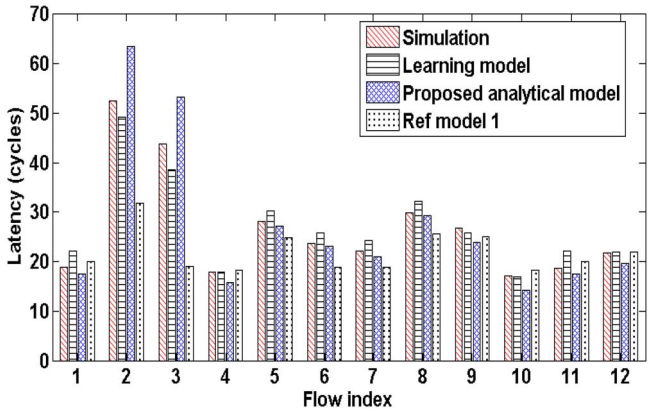

Fig. 17. Latency comparisons of each flow in the consumer application of E3S benchmark [36]. The four bars correspond to the simulation, learning model, proposed analytical model, and reference model 1 [6], respectively.

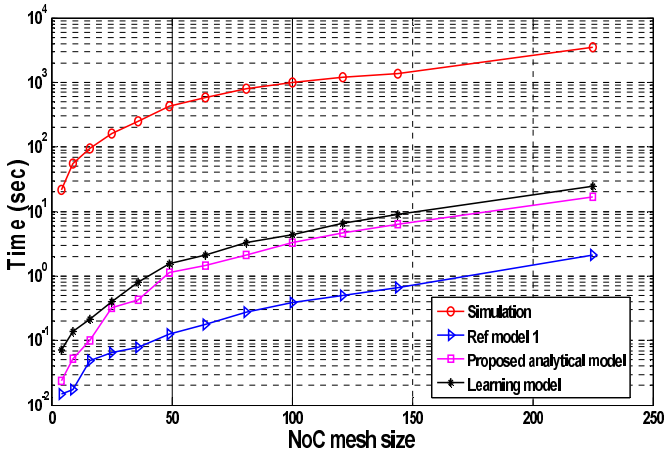

Fig. 18. Run time comparisons of the SVR-NoC learning model, proposed analytical model, reference model 1 [6], and simulation under different mesh network sizes. The reference model 1 is implemented in MATLAB based on the derivation presented in [6].

prediction takes only a few seconds across different traffic. A more than 100× speedup is achieved.

In Fig. 18, we compare the run times of the simulation approach, the proposed analytical model, the reference model 1 [6], as well as the learning models under different mesh sizes for the same traffic input. The reference model 1 achieves the fastest speed as it only uses a close-form G/G/1 formula for obtaining the mean waiting time [6]. On the other hand, the proposed analytical model needs to solve the state distribution equations for the queuing systems as in [11] and [20]. Also, it can be observed the overhead of applying the regression models in SVR-NoC is not large. In summary, the proposed learning framework can use any other analytical models as features provided the model is suitable for the target platform. Therefore, for NoCs with priority-based arbitration, one can also embed predictions from Ref model 1 into the feature vectors to accelerate the design space exploration.

Also from Fig. 18, it is observed the run times of both analytical models and learning models grow almost linearly with the NoC size. This is because both learning and analytical models perform analysis on the LDG nodes. Since the number of link channels in LDG grows linearly with NoC size, a corresponding increasing in run time is observed.

VI. Conclusion

In this paper, we have proposed a learning-based approach for NoC average latency prediction. We first proposed an analytical model with two equivalent queuing systems to derive the path contention time and the transfer time, respectively. Then, we proposed a learning framework to improve the estimation accuracy. We have explored the features that are required to effectively learn the source waiting time and the channel queuing delay models. Compared to the analytical queueing models, the SVR-NoC model can improve the prediction accuracy for most traffic patterns. As a future work, we plan to explore a learning model with on-line learning capability to further improve the prediction accuracy over more traffic patterns.

Acknowledgment

The authors would like to thank anonymous reviewers for their suggestions and comments.

References

[1] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[2] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Automat. Conf.*, Las Vegas, NV, USA, 2001, pp. 684–689.
[3] U. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 12, pp. 2001–2013, Dec. 2010.
[4] A. E. Kiasari, A. Jantsch, and Z. Lu, "Mathematical formalisms for performance evaluation of networks-on-chip," *ACM Comput. Surv.*, vol. 45, no. 3, pp. 38:1–38:41, Jul. 2013.
[5] M. Lai, L. Gao, N. Xiao, and Z. Wang, "An accurate and efficient performance analysis approach based on queuing model for network on chip," in *ICCAD Dig. Tech. Papers*, San Jose, CA, USA, Nov. 2009, pp. 563–570.
[6] A. E. Kiasari, Z. Lu, and A. Jantsch, "An analytical latency model for networks-on-chip," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 1, pp. 113–123, Jan. 2013.
[7] Z. Guz *et al.*, "Network delays and link capacities in application-specific wormhole NoCs," *VLSI Design*, vol. 2007, Feb. 2007, Art. ID 90941.
[8] E. Fischer and G. Fettweis, "An accurate and scalable analytic model for round-robin arbitration in network-on-chip," in *Proc. 7th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, Tempe, AZ, USA, Apr. 2013, pp. 1–8.
[9] P. Bogdan and R. Marculescu, "Non-stationary traffic analysis and its implications on multicore platform design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 508–519, Apr. 2011.
[10] Y. Wu, G. Min, M. Ould-Khaoua, H. Yin, and L. Wang, "Analytical modelling of networks in multicomputer systems under bursty and batch arrival traffic," *J. Supercomput.*, vol. 51, no. 2, pp. 115–130, 2010.
[11] P.-C. Hu and L. Kleinrock, "An analytical model for wormhole routing with finite size input buffers," in *Proc. 15th Int. Teletraff. Congr.*, Washington, DC, USA, 1997, pp. 549–560.
[12] N. Nikitin and J. Cortadella, "A performance analytical model for network-on-chip with constant service time routers," in *ICCAD Dig. Tech. Papers*, San Jose, CA, USA, Nov. 2009, pp. 571–578.
[13] Z. Qian *et al.*, "A comprehensive and accurate latency model for network-on-chip performance analysis," in *Proc. 19th Asia South Pac. Design Automat. Conf. (ASP-DAC)*, Singapore, Jan. 2014, pp. 323–328.
[14] Y. Ben-Itzhak, I. Cidon, and A. Kolodny, "Delay analysis of wormhole based heterogeneous NoC," in *Proc. 5th IEEE/ACM Int. Symp. Netw. Chip (NoCS)*, Pittsburgh, PA, USA, 2011, pp. 161–168.
[15] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook," *Perform. Eval.*, vol. 18, no. 2, pp. 149–171, 1993.

[16] C. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.
[17] V. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.
[18] A. Kahng, B. Lin, and K. Samadi, "Improved on-chip router analytical power and area modeling," in *Proc. 15th Asia South Pac. Design Automat. Conf. (ASP-DAC)*, Taipei, Taiwan, 2010, pp. 241–246.
[19] Z. Qian *et al.*, "SVR-NoC: A performance analysis tool for network-on-chips using learning-based support vector regression model," in *Proc. Design Automat. Test Europe Conf. Exhibit. (DATE)*, Grenoble, France, Mar. 2013, pp. 354–357.
[20] D. D. Kouvatsos, S. Assi, and M. Ould-Khaoua, "Performance modeling of wormhole-routed hypercubes with bursty traffic and finite buffers," *Int. J. Simul. Pract. Syst. Technol.*, vol. 6, nos. 3–4, pp. 69–81, 2005.
[21] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann, 2003.
[22] E. Krimer, I. Keslassy, A. Kolodny, I. Walter, and M. Erez, "Static timing analysis for modeling QoS in networks-on-chip," *J. Parallel Distrib. Comput.*, vol. 71, no. 5, pp. 687–699, 2011.
[23] M. Arjomand and H. Sarbazi-Azad, "Power-performance analysis of networks-on-chip with arbitrary buffer allocation schemes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 29, no. 10, pp. 1558–1571, Oct. 2010.
[24] S. Foroutan, Y. Thonnart, and F. Petrot, "An iterative computational technique for performance evaluation of networks-on-chip," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1641–1655, Aug. 2013.
[25] (2014). *MATLABBGL*. [Online]. Available: https://www.cs.purdue.edu/homes/dgleich/packages/matlab_bgl/
[26] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. New York, NY, USA: McGraw-Hill, 2001.
[27] L. Kleinrock, *Queueing Systems, Volume I: Theory*. New York, NY, USA: Wiley, 1975.
[28] P. J. Kuhn, "Tutorial on queuing theory," Univ. Stuttgart, Stuttgart, Germany, 2013.
[29] A. Smola and B. Scholkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.
[30] H.-K. Peng, C.-P. Wen, and J. Bhadra, "On soft error rate analysis of scaled CMOS designs—A statistical perspective," in *IEEE/ACM ICCAD Dig. Tech. Papers*, San Jose, CA, USA, Nov. 2009, pp. 157–163.
[31] (2012). *Libsvm*. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm
[32] (2012). *Lssvm*. [Online]. Available: http://www.esat.kuleuven.be/sista/lssvmlab/
[33] (2012). *Booksim 2.0*. [Online]. Available: http://nocs.stanford.edu/booksim.html
[34] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 4, pp. 551–562, Apr. 2005.
[35] D. Bertozzi *et al.*, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.
[36] (2010). *E3S Benchmark Suite*. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/
[37] (2000). *SPEC Benchmark*. [Online]. Available: http://www.specbench.org/
[38] S. Ma, N. Jerger, and Z. Wang, "DBAR: An efficient routing algorithm to support multiple concurrent applications in networks-on-chip," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, San Jose, CA, USA, Jun. 2011, pp. 413–424.
[39] M. Palesi and M. Daneshtalab, Eds., *Routing Algorithms in Networks-on-Chip*. New York, NY, USA: Springer, 2014.

Zhi-Liang Qian received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2008, and the Ph.D. degree in electronic and computer engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2014.

His current research interests include high-performance network-on-chip design, low-power very-large-scale integration implementation, and embedded system design.

Da-Cheng Juan received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2014.

He is currently with Google Inc., Mountain View, CA, USA, developing machine-learned recommendation systems. His current research interests include applied machine learning, energy-efficient computing, and algorithmic programming.

Dr. Juan was a recipient of several awards from major programming contests.

Paul Bogdan received the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2011.

He is currently an Assistant Professor with the Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA. His current research interests include performance analysis and design methodologies for multicore systems, cyber-physical systems, and modeling and analysis of biological systems.

Chi-Ying Tsui (SM'11) received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1994.

He joined the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, in 1994, where he is currently a Full Professor. He has published over 180 referred publications and holds ten U.S. patents. His current research interests include designing very-large-scale integration architectures for low-power applications, developing power management circuits for embedded portable devices.

Prof. Tsui was a recipient of the Best Paper Awards from the IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION (VLSI) SYSTEMS in 1995, the IEEE International Symposium on Circuits and Systems in 1999, the IEEE/ACM ISLPED in 2007, the IEEE International Symposium on Electronic Design, Test & Applications in 2008, and the International Conference on Hardware/Software Codesign and System Synthesis in 2012, and the Design Awards in the IEEE Asia and South Pacific Design Automation Conference (ASP-DAC) University Design Contest in 2004 and 2006.

Diana Marculescu (F'15) received the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1998.

She is currently with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. Her current research interests include energy- and reliability-aware computing, and CAD for nonsilicon applications, such as computational biology and sustainability.

Dr. Marculescu was a recipient of the National Science Foundation Faculty Career Award from 2000 to 2004, the ACM Special Interest Group on Design Automation Technical Leadership Award in 2003, the Carnegie Institute of Technology George Tallman Ladd Research Award in 2004, and the several Best Paper Awards at the IEEE ASP-DAC in 2005, the IEEE International Conference on Computer Design in 2008, the IEEE International Symposium on Quality Electronic Design in 2009, and the IEEE TRANSACTIONS ON VERY LARGE-SCALE INTEGRATION (VLSI) SYSTEMS in 2011. She is an ACM Distinguished Scientist.

Radu Marculescu (F'13) received the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 1998.

He is a Professor with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA.

Prof. Marculescu was a recipient of several Best Paper Awards in top conferences and journals covering design automation of integrated systems and embedded systems. His current research interests include modeling and optimization of embedded systems, cyber-physical systems, and social and biological systems.